ECE 6882 – Reinforcement Learning

Spring 2023 – The George Washington University

Osama Yousuf – osamayousuf@gwu.edu

Project 3 – Solving a Maze using SARSA and Q-learning Algorithms

1. Introduction

In this project, I have implemented two different algorithms, namely SARSA and Q-learning, to solve the provided maze problem. I have carried out experiments to establish a sense of which algorithm performs better. Similar to Project 2, I used environment 4 and edited its functionality to my liking. Supplemental code files are provided as an accompanying archive. Table 1 highlights the purpose of each file.

Table 1: Description of implemented code.

File	Description
main.py	Entry point. Executing this as is will replicate all plots used in this report.
policies.py	Contains my implementations of the SARSA and QLearning policies. Policies from Project 2 are also included.
util.py	Contains environment-specific functionality such as classes for the maze and agent, as well as visualization functions.
mazes/base.txt	This is the base file for the 18 x 18 maze which we have to solve for this project. This is directly used in main.py. This could be replaced with any other maze directly, and the environment should correctly load and solve it using both policy iteration as well value iteration.

2. Environment Description

As highlighted in Table 1, my environment can load a maze for solving using generic text files, as long as the mapping is respected. Figure 1 shows the base maze, which contains a total of 18 x 18 possible states.

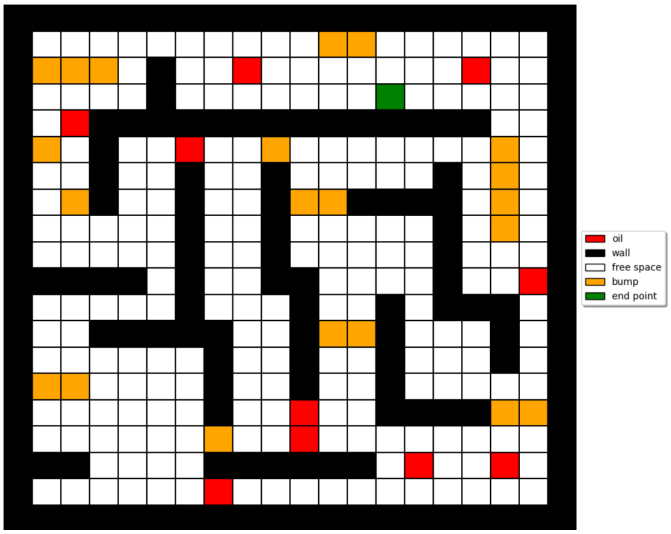

Figure 1: Base maze environment loaded from mazes/base.txt

2.1 State Space (S): The state space contains all possible cells of the 18 x 18 matrix with the exception of walls where the agent can be present at any given point. Each state can either be empty, full (walled off), bump, oil, or goal. Accounting for walls, the total number of states are $18 \times 18 - 76 = 248$.

2.2 Action Space (A): An agent can take one of four possible actions at any given state. These are: Up, Down, Left, and Right. In my environment, these are coded as action 0, 1, 2, and 3 respectively.

2.3 Transition probabilities (T): When an action is chosen, the agent can either move to one of its neighbor states or stay in the same sell depending on p – the transition randomness parameter – and whether the anticipated state is a wall or not. In exact terms, after taking any given action, the agent moves to the anticipated state with a probability of $1 - p$ and moves to one of the other neighboring cells with a probability of $p/3$. If the state to transition to is a wall, then the agent simply stays in the current cell. p can take on values between 0 and 1 (inclusive), where $p = 0$ means that the agent will always transition to its intended state and $p = 1$ means that the agent will always transition to an unintended neighboring state with equal probability of $1/3$.

2.4 Reward Function: The agent receives a -1 reward for all actions regardless of the state it transitions to. It also receives additional special rewards if it transitions to oil, bump, and goal states, as specified in Table 2.

State (after transition)	**Reward**
All actions (Up, Down, Left, Right)	-1
Oil	-5
Bump	-10
Goal	+200

Table 2: Reward as a function of transitioned state

3. Algorithmic Comparison: SARSA vs. Q-learning

SARSA (State-Action-Reward-State-Action) and Q-learning are both reinforcement learning algorithms, but they both have subtleties and differences that differentiate their performance from one another.

- **SARSA** is an **on-policy** learning algorithm, meaning that the agent learns from experiences generated by following the current policy. The same policy is used to generate actions as well as update the policy. **Q-learning** on the other hand is an **off-policy** algorithm, where the agent learns from the experiences generated by following a different policy (greedy) than the one currently being used. Different policies are used to generate actions and update the policy.
- SARSA learns the Q-value based on the next state and action taken by the agent, whereas Q-learning simply learns Q-value based on the action that gets the maximum possible reward from the next state. This is evident in the update step used in these two algorithms. For SARSA, the update is $Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$. For Q-learning, the update is $Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$. The different hyperparameters in these equations are summarized in Section 4.
- It is known that SARSA, owing to its on-policy nature, can converge to a suboptimal policy in a few cases. On the other hand, Q-learning is guaranteed to converge to an optimal policy.

Compared to policy iteration and value iteration algorithms (which were model-based), both SARSA and Q-learning are model-free algorithms. This means that they do not require explicit knowledge of the underlying transition probabilities for learning the optimal policy.

4. Algorithmic Hyperparameters

Table 3 summarizes the exact values used for the different algorithmic hyperparameters in my simulations, followed by an explanation of what each hyperparameter corresponds to.

Hyperparameter	Value
p	0.02
γ	0.95
α	0.3
ϵ	0.1
Episodes (E)	1000
Episode Steps (ES)	1000
Policy Steps (PS)	500
Runs (R)	10

Table 3: Summary of chosen algorithmic hyperparameters

- p is the transition randomness of the agent (as explained in Section 2.3). The agent transitions to an intended state with a probability of $1 - p$, and a random neighbor with equal probability $p/3$. Higher the value of p (between 0 and 1, inclusive), higher is the randomness in the agent in following an intended trajectory.
- γ is the discount factor allotted to future rewards. It directly represents the importance of future rewards relative to immediate rewards. The higher the value (between 0 and 1, inclusive), the more important future rewards are considered relative to the immediate reward.
- α is the learning rate that is multiplied by the error term in the Q-value update equation for each algorithm (given in Section 3 point 2).
- ϵ is the parameter for the ϵ-greedy policy that is derived from the Q-values. For SARSA, ϵ-greedy is used when the agent has to choose the initial as well as next state actions. For Q-learning, it is used for the initial action. ϵ-greedy means that a greedy action (one which maximizes the Q-value) is chosen with a probability of $1 - \epsilon$; otherwise, an action is drawn randomly with equal probability over all possible actions.
- E refers to the total number of episodes for the SARSA and Q-learning algorithms.
- ES refers to the total number of steps an agent is allowed to take within each episode. Once ES steps have been taken, an episode is concluded.
- PS refers to the total number of steps an agent can take to reach the goal state. If an agent does not reach the goal state within PS steps, the policy is classified to have failed in finding a path from the start to a goal state. If an agent reaches the goal state within PS steps, the policy is classified to have successfully found a path from the start to a goal state.
- R represents the total number of independent runs for which each algorithm is repeated. This is important so as to capture statistically significant results pertaining to the performance of each algorithm.

Apart from these hyperparameters, it is also important to note how tie-cases are handled in my implementation. A tie can occur when more than one action leads to the optimal Q-value. In such a case, competing actions are chosen randomly with equal probability during the learning phase. After the Q-values have been learned, the optimal policy is derived directly by choosing the action with the highest Q-value for each state. If there's a tie in this phase – it is broken simply by choosing

the first action with the highest Q-value (in the sequence Up, Down, Left, Right). For reproducibility, a random seed is fixed in my simulations.

5. Results

a) Convergence:

Both the SARSA and Q-learning algorithms were executed for a total of 10 independent iterations. Out of these, the **SARSA algorithm** was able to find a path from the start to the goal state **7 times**, whereas the **Q-learning algorithm** found a path from the start to goal state **10 times**. This means that under the chosen set of hyperparameters, the Q-learning algorithm exhibited better performance in terms of solving the underlying problem relative to the SARSA algorithm. This is also in alignment with the comparison presented in Section 3 point 3, where I discussed that the SARSA algorithm can converge to suboptimal solutions in a few cases, but the Q-learning algorithm is guaranteed to converge to an optimal solution. It is still entirely possible that the SARSA algorithm's convergence performance can be increased by optimizing algorithmic hyperparameters.

It must be noted that the current results are limited to the same starting position in the maze. For fully capturing convergence-level details, one must evaluate and compare the performance of the two algorithms (whether the agent reaches the goal state or not) over all possible starting states. This is left as future work.

b) Optimal Policy & Path:

After learning has terminated, we are left with Q-values for each action for each state in the state space. The optimal policy can be directly extracted using these Q-values as follows: For a given state s, the optimal policy is given by a^*, where $a^* = argmax_a\, Q(s,a) \quad \forall a$.

Figures 2 and 3 show the optimal policies learned by the SARSA algorithm and Q-learning algorithms respectively for one of the 10 independent runs. It can be seen that there are differences in the two, which can primarily be attributed to the differences in the underlying Q-value update equations described in Section 3 point 2.

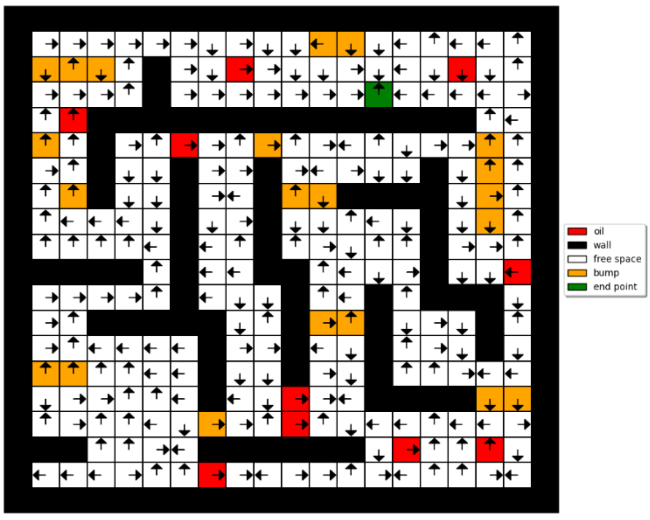

Figure 2: Optimal policy learned by SARSA algorithm (run = 10). Algorithmic parameters are described in Table 2.

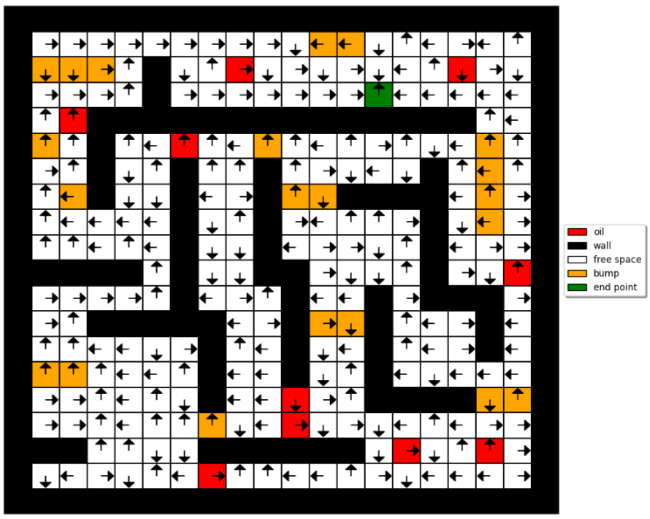

Figure 3: Optimal policy learned by Q-learning algorithm (run = 1). Algorithmic parameters are described in Table 2.

Figures 4 and 5 show the optimal path obtained by the SARSA algorithm and Q-learning algorithms respectively for one of the 10 independent runs. Despite the fact that the agent's trajectories are different across the two algorithms, it can be seen that the agent takes the same number of steps (37) and achieves the same total reward (158). Thus, it can be concluded that for the given starting state and independent run, both algorithms solve the underlying maze problem successfully.

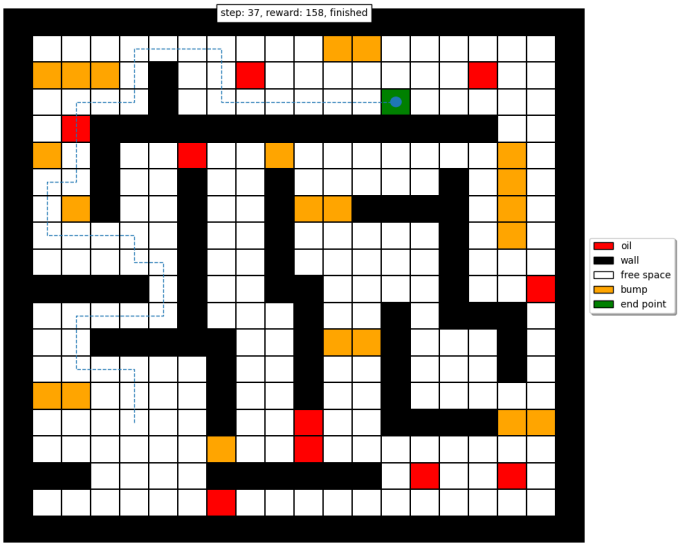

Figure 4: Optimal path obtained by SARSA algorithm (run = 10). Algorithmic parameters are described in Table 2.

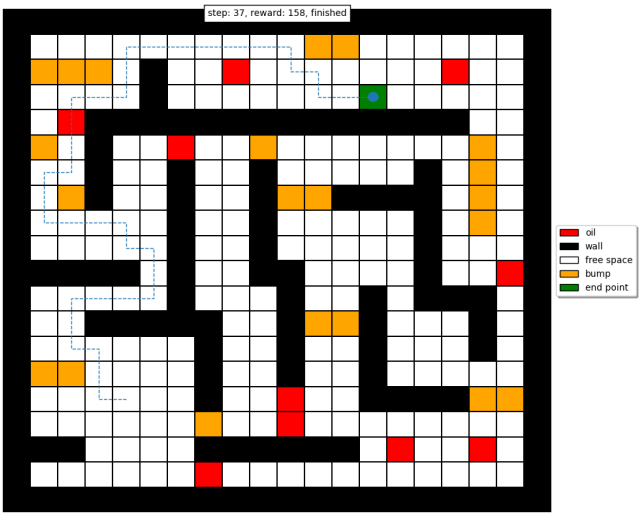

Figure 5: Optimal path obtained by Q-learning algorithm (run = 1). Algorithmic parameters are described in Table 2.

c) *Average Accumulated Reward:*

The average accumulated reward is determined by calculating the accumulated rewards for each independent run as follows:

$$AccR_k^i = \frac{1}{k}\sum_{j=1}^{k} r_j \,,$$

where $AccR_k^i$ denotes the average reward per step obtained by the agent up to the time step k in the $i-$ th independent run. Then, the average over R independent runs of accumulated rewards is calculated as:

$$\overline{AccR_k} = \frac{1}{R}\sum_{j=1}^{R} AccR_k^i$$

Figures 6 and 7 show the average accumulated reward $\left(\overline{AccR_k}\right)$ with respect to the episode for the SARSA and Q-learning algorithms respectively. Figure 8 presents the same information together for better comparison. It can be seen that both algorithms achieve a higher accumulated reward as episodes increase. This indicates that the Q-values generally improve in a direction that leads to the agent receiving higher rewards and thus performing better each episode. Moreover, the Q-learning algorithm consistently outperforms the SARSA algorithm in terms of the average accumulated reward with respect to learning episode, either attaining equally good or better accumulated rewards per step. For the 1000^{th} episode, Q-learning has an accumulated reward of 3.0, whereas SARSA has an accumulated reward of 2.4.

This difference can be attributed to the fact that SARSA learns a close-to-optimal policy while performing exploration, whereas Q-learning directly learns the optimal policy via off-policy greedy actions. Having the ϵ parameter either be statically smaller or dynamically attenuating over time could assist the SARSA algorithm in achieving comparable performance to Q-learning in terms of the average accumulated reward.

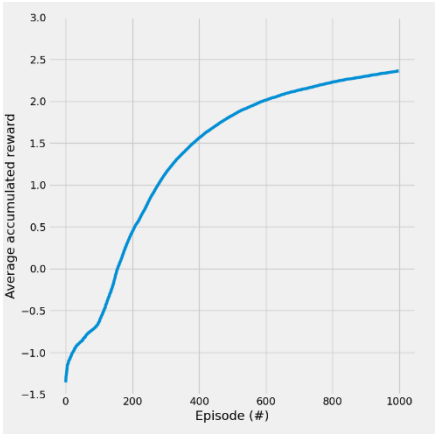

Figure 6: Average accumulated reward w.r.t. episode of the SARSA algorithm over 10 independent runs. Other algorithmic parameters are described in Table 2.

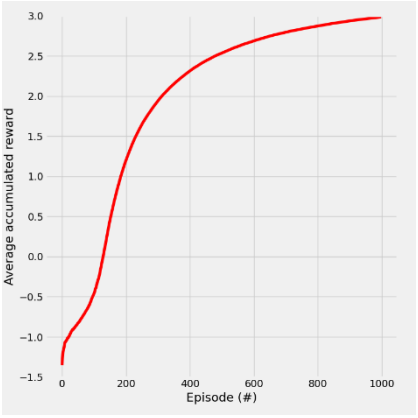

Figure 7: Average accumulated reward w.r.t. episode of the Q-learning algorithm over 10 independent runs. Other algorithmic parameters are described in Table 2.

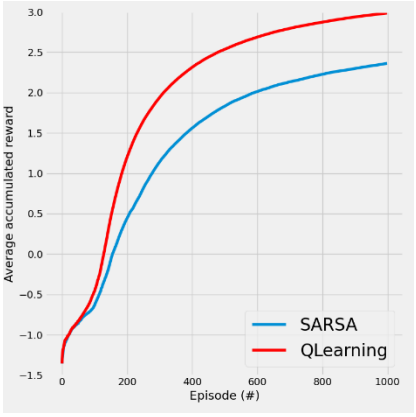

Figure 8: Comparison of the average accumulated reward w.r.t. episode of the SARSA and Q-learning algorithms over 10 independent runs. Other algorithmic parameters are described in Table 2.

d) Average Time to Converge:

I also calculated the time to converge for both the algorithms, averaged over the 10 independent runs. My simulations show that on average, the time to converge for Q-learning is 8 seconds, and the same for SARSA is 16 seconds. This means that Q-learning consistently converges twice as fast compared to SARSA.

This faster time to converge can be explained by looking at the learning dynamics and accumulated reward curves from Figure 8. Since Q-learning greedily chooses the action with the max reward, it gets to the goal state in lesser steps compared to SARSA. Once the goal state is reached, an episode is directly terminated. Thus, the faster convergence times of Q-learning are due to the episodes terminating early.

Furthermore, in the current problem, we know that all states except the goal state lead to a negative reward. This means that for any given episode, a policy that reaches the goal state in lower number of steps will accumulate a higher reward than a policy that reaches the goal state in a higher number of steps. Since SARSA performs more exploration and has a longer time to converge, it is implied that it would also receive relatively lower rewards compared to Q-learning, and this is directly evident from Figure 8.

6. Conclusion

In this project, I studied the SARSA and Q-learning reinforcement learning algorithms and explored their performance for a simple maze problem. Results show that a) Q-learning exhibits a higher successful convergence rate compared to SARSA (100 % against 70 %), b) Q-learning achieves a higher average accumulated reward with respect to learning episode compared to SARSA (3.0 against 2.4), and c) on average, Q-learning converges twice as fast compared to SARSA (requiring 8 seconds against SARSA's 16 seconds). Future work can explore attenuating ϵ values for the SARSA algorithm in order to bring its performance on par with Q-learning.